Week 9 : Pandas

pending tasks	
type	

Introduction - pandas

Missing features in numpy to enable relational tables:

- No way to attach labels to the data.
- Missing pre-built methods to fill missing values.
- No way to group data.
- No way to pivot data.

Pandas is built on top of numpy to make rational data processing easier.

Creating series object

- A list of whole numbers is created implicitly as the index of the series. All the values of a series object are of the same data type.
- For a given series object s s.values gives all the values and s.index gives all the index values that can be iterated over. The individual values can be accessed using the index, e.g. s[2] is the third object of s.

```
# creating from lists
s = pd.Series([0, 1, 1 ,2 ,3 ,5 ,8])
```

- When the index is specified, the object values can be accessed using the given index.

```
#specifying index
mercury = pd.Series([0.33, 57.9, 4222.6], index = ['mass', 'diameter', 'dayLength'])
mercury['mass'] # not advised to use mars.mass
>>> 0.33
```

- Arrays can also be used to create series objects.

```
# creating series from an array
arr = np.random.randint(0, 10, 10)
ind = no.arange(10,20)
rand_series = pd.Series(arr, index = ind)
```

- Series can be created using dictionaries. Here apart from the index in the dictionary, an explicit index can be specified in the series object. Only the mentioned indices will be read from the dictionary to the series object.

```
#creating series from a dictionary
d = {}
d['mass'] = 0.33
d['diameter'] = 57.9
d['dayLength'] = 4222.6
mercury = pd.Series(d, index = ['mass', 'diameter'])
```

iloc and loc - indexing series objects

- A series object can have two indices : an explicit index defined by the user and an implicit index created based on the object position. iloc and loc are used for this purpose.
- loc is used to access attributes using the explicit index and iloc is used to access the elements using the implicit index. An example for this is given below, where the fourth element is accessed using loc and iloc. loc takes the explicit index 4 (starting from 1) whereas iloc takes the implicit index 4 (starting from 0). It is a good practice to use

loc and iloc rather than subscripting directly using s[4]. Slicing can be used in both, example of mercury series is given below.

```
# using loc and iloc
s = pd.Series([0.0, 1, 1 ,2 ,3 ,5 ,8], index = [1, 2, 3, 4, 5, 6, 7])
s.loc[4]
>>> 2.0
s.iloc[4]
>>> 3.0
# accessing elements of mercury series
mercury.iloc[0]
>>> 0.33
mercury.loc['mass']
>>> 0.33
mercury.iloc[0:2]
>>> mass       0.33
    diameter  57.90
    dtype : float64
mercury.loc['mass':'diameter']
>>> mass       0.33
    diameter  57.90
    dtype : float64
```

Simple operations

- Slices of series objects can be made by giving conditional statements instead of positions. In the example of mass Series, a condition of mass > 100 returns a boolean value for all the indices. This can be used to return another series object as shown.

```
mass = pd.Series([0.33, 4.87, 5.97, 0.642, 1898, 568, 86.8, 102, 0.0146],
                 index=['Mercury', 'Venus', 'Earth', 'Mars', 'Jupiter', 'Saturn', 'Uranus', 'Neptune', 'Pluto'])
mass[(mass > 100) & (mass < 600)]
>>> Saturn     568.0
    Neptune    102.0
    dtype: float64
```

- All the standard numpy functions can be used on Series objects, this returns a new series object. It also supports using numpy objects like np.mean(series_object).
- When adding two series objects, they are aligned based on the indices and then added. Therefore, the result would include values only for the intersecting index labels of the two series. The other indices of union will be **NaN** (not a number) values.
- pd.isnull(series_obj) returns boolean values True for null and false for not null.

```
new_mass[~pd.isnull(new_mass)]
>>> Jupiter    3796.0
    Neptune     204.0
    Saturn     1136.0
    dtype: float64
```

- A new row can be added by specifying the new index and assigning it a value. series_obj.drop[index] can be used to drop values from the series.

```
# adding a new element to the series
mass['Moon'] = 0.7346
# removing an element from the series
mass.drop(['Pluto'])
```

Question1:

Collect numbers for the diameters of these planets (heavenly bodies) and store it as a Series object. Then for the given Series objects mass and diameter, compute the density of each planet.

solution notes :

Since series are numpy arrays with indices, pointwise operations are possible. For indices that do not align, NaN value is assigned.

```
density = mass / (np.pi * np.power(diameter, 3) / 6)
```

Question2:

Given this density Series, replace all values which NaNs with the mean density of all planets.

solution notes :

```
density[pd.isnull(density)] = np.mean(density)
```

Question3:

Compare dictionary with series : checking presence of key, sum of values, computing std.

solution notes:

- Dictionaries are almost two times faster in checking membership.

```
%%timeit
for i in arr:
    i in my_dict
>>>  100 loops, best of 3: 6.73 ms per loop
%%timeit
for i in arr:
    i in my_series
>>>  100 loops, best of 3: 10.4 ms per loop
```

- All the other operations are significantly faster while using series object.

NIFTY case study

- Series objects align indices before any mathematical operations. Thus, use series_obj.values().
- The tasks two and three focus on processing the index column.

Summary

- Pandas is built over numpy to facilitate relational data processing. There are two types of pandas objects namely, series and dataframes.
- A series object can be initialized using a list, array or dictionary. loc and iloc are used to index the pandas objects which use explicit and implicit indexing respectively.
- Series objects are added by aligning the indices, thus, the result includes only the intersecting indices.

MCQ : Week 9

1. Pandas is used to
    1. create a GUI for data representation.
    2. **create a high level array.**
    3. create a database.
    4. None of the above.
2. Data can be analysed using
    1. Pandas series objects
    2. Pandas dataframe objects
    3. **Both A and B**
    4. Only numpy can be used for meaningful data analysis.
3. An operation between unaligned Series will have the _____ of the indexes involved.

1. **Union**
2. Intersection
3. All the indices from the first series
4. All of the above

4. Series is a one-dimensional labeled array capable of holding any data type.
    1. **True**
    2. False
5. From the given series s, which statement will replace the TODO

```
s = pd.Series([0.0, 1, 1, 2, 3, 5, 8], index=[1, 2, 3, 4, 5, 6, 7])
# TODO : Print "2.0 3.0" from the series
>>> 2.0  3.0
```

    1. **print(s.loc[4],s.iloc[4])**
    2. **print(s.iloc[3],s.iloc[4])**
    3. print(s.loc[5],s.iloc[3])
    4. print(s.loc[4],s.iloc[3])